Zellic

LayerZero Wrapped Asset Bridge

Smart Contract Security Assessment

February 20, 2023

Prepared for:

Ryan Zarick

LayerZero Labs

Prepared by:

Katerina Belotskaia and Vlad Toie

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for LayerZero Labs from February 16th to February 17th, 2023. During this engagement, Zellic reviewed LayerZero Wrapped Asset Bridge's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could an on-chain attacker drain locked tokens?
- Could an on-chain attacker manipulate the amount or address of receiving tokens?
- Could a user's funds be forever locked without possibility to transfer over the bridge?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Problems relating to the front-end components and infrastructure of the project
- Problems due to improper key custody or off-chain access control
- Issues stemming from code or infrastructure outside of the assessment scope

1.3 Results

During our assessment on the scoped LayerZero Wrapped Asset Bridge contracts, we discovered one low severity finding. No critical issues were found.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	0

2 Introduction

2.1 About LayerZero Wrapped Asset Bridge

LayerZero Wrapped Asset Bridge allows bridging ERC20 tokens and native gas tokens (e.g., ETH) from existing EVM chains (e.g., Ethereum, Avalanche, BSC) to subnets or brand-new EVM chains where those assets do not exist natively. It supports mapping the same wrapped token to multiple tokens on other chains. For example, moving native USDC from Ethereum or Avalanche to NewChainX will result in the same wrapped asset on NewChainX.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We

look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

LayerZero Wrapped Asset Bridge Contracts

Repository	https://github.com/LayerZero-Labs/wrapped-asset-bridge
Versions	`34b787e9ed6b9e4338a9c200a656bff65e016b52`
Programs	• OriginalTokenBridge • TokenBridgeBase • WrappedERC20 • WrappedTokenBridge
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-days. The assessment was conducted over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-founder
jazzy@zellic.io

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

Vlad Toie, Engineer
vlad@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

February 16, 2023	Start of primary review period
February 17, 2023	End of primary review period

3 Detailed Findings

3.1 The contract owner can change already registered token addresses

- **Target**: WrappedTokenBridge
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The owner of the contract can register a pair of token addresses over the `registerToken` function. These token addresses are used to determine which token should be minted when received from the remote chain and which token will be burned, and the corresponding token will be received in the remote chain during transfer over bridge.

```
function registerToken(address localToken, uint16 remoteChainId,
    address remoteToken) external onlyOwner {
        require(localToken ≠ address(0), “WrappedTokenBridge: invalid
    local token”);
        require(remoteToken ≠ address(0), “WrappedTokenBridge: invalid
    remote token”);

        localToRemote[localToken][remoteChainId] = remoteToken;
        remoteToLocal[remoteToken][remoteChainId] = localToken;
        emit RegisterToken(localToken, remoteChainId, remoteToken);
    }
```

But this function also allows the owner to change previously registered address pairs.

Impact

As a result of changing an already registered pair of addresses, a situation may occur in which one local token will be minted, and after reregistering the addresses, another one will be burned to receive the same remote token. Also, the same remote token address can be related to two local token addresses and vice versa.

Recommendations

Add a check that the tokens' addresses have already been registered before.

Remediation

LayerZero Labs acknowledged this finding and implemented a fix in commits 5b3fd60f and fee46f08.

4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1 Module: `WrappedTokenBridge.sol`

Function: `registerToken()`

Allows the owner of the contract to add related local and remote token addresses.

Branches and code coverage

Intended branches:

- Addresses for the remoteChainId are correctly set.
  ☑ Test coverage

Negative behavior:

- Tokens already registered.
  ☑ Negative test?
- The caller is not the owner.
  ☑ Negative test?
- One of the addresses is 0.
  ☑ Negative test?

Inputs

- `remoteToken`:
  - **Control**: Only owner has full control.
  - **Constraints**: Is not a zero address.
  - **Impact**: The address of the contract from another chain.
- `remoteChainId`:

  - **Control**: Only owner has full control.
  - **Constraints**: No checks.
  - **Impact**: The trusted chainId.
- `localToken`:
  - **Control**: Owner only has full control.
  - **Constraints**: Is not a zero address.
  - **Impact**: The address of the contract to which tokens will be received from another chain. Also, it will be possible to send tokens from this address.

Function: `setWithdrawalFeeBps()`

Allows the owner of the contract to set the fee BPS value.

Branches and code coverage

Intended branches:

- The new value is correctly set.
  ☑ Test coverage

Negative behavior:

- The new value is more than `TOTAL_BPS`.
  ☑ Negative test?
- The caller is not the owner.
  ☑ Negative test?

Inputs

- `_withdrawalFeeBps`:
  - **Control**: Full control.
  - **Constraints**: Less than `TOTAL_BPS`.
  - **Impact**: The value of the fee that will be charged for sending tokens back to the original chain.

Function: `bridge()`

Allows to send the tokens to the remote chain. The remote token address is not controlled by the caller and can only be set by the contract owner. The sent tokens will be burned.

Preconditions

The tokens from `remoteToken` were received previously, the `localToken` is registered, and the `msg.sender` has `amount` value of `localToken`.

Branches and code coverage

Intended branches:

- The global token counter for remote token address should decrease.
  ☑ Test coverage
- The balance of `localToken` of the caller should be decreased by `amount` value.
  ☑ Test coverage

Negative behavior:

- The caller doesn't have enough tokens.
  ☑ Negative test?
- The `localToken` is not registered.
  ☑ Negative test?

Inputs

- `adapterParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Value is used to determine the amount of produced gas limit.
- `callParams`(refundAddress and zroPaymentAddress):
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: No impact.
- `unwrapWeth`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: If `remoteToken` is wETH and `unwrapWeth` is true, the `to` address will receive the native tokens.
- `to`:
  - **Control**: Full control.
  - **Constraints**: `to` $\neq$ `address(0)`.
  - **Impact**: The receiver of the tokens on the remote chain side.

- `amount`:
  - **Control**: Full control.
  - **Constraints**: The caller's balance and `totalValueLocked[remoteChainId][remoteToken]` should be more or equal to the `amount` value, `amount > 0`.
  - **Impact**: The number of local tokens that will be burned.
- `remoteChainId`:
  - **Control**: Full control.
  - **Constraints**: The `remoteChainId` should be registered.
  - **Impact**: Caller shouldn't be able to send the tokens to the unknown chain.
- `localToken`:
  - **Control**: Full control.
  - **Constraints**: `remoteToken` for this `localToken` is not zero.
  - **Impact**: The tokens will be burned for receiving the original tokens on the remote chain side.

Function call analysis

- `IWrappedERC20(localToken).burn(msg.sender, amount)`
  - **What is controllable?** `localToken`, `msg.sender`, and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The function has a `nonReentrant` modifier. The function will revert if the balance of `msg.sender` is less than `amount` value.
- `_lzSend(remoteChainId, payload, callParams.refundAddress, callParams.zroPaymentAddress, adapterParams, msg.value)` -> `lzEndpoint.send{value: _nativeFee}(_dstChainId, trustedRemote, _payload, _refundAddress, _zroPaymentAddress, _adapterParams)`
  - **What is controllable?** `remoteChainId`, `refundAddress`, `zroPaymentAddress`, `adapterParams`, `msg.value`
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Will revert if `remoteChainId` is unknown, if `msg.value` is not enough to pay the fee, or if the `_zroPaymentAddress` address is not the contract address or is not `tx.origin` (in case of fee in ZRO token). Also, it will revert in case of error during refund of native tokens to `refundAddress`.

Function: `_nonblockingLzReceive()`

The internal function called from the `lzReceive` function. The `lzReceive` can be called only by a trusted `lzEndpoint` address.

Branches and code coverage

Intended branches:

- The global token counter for remote token address should increase.
  ☑ Test coverage
- The balance of `localToken` of the `to` address should be increased by `amount` value.
  ☑ Test coverage

Negative behavior:

- Incorrect packet type.
  ☑ Negative test?
- The `remoteToken` is not registered.
  ☑ Negative test?

Inputs

- `amount`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The amount of `localToken` that will be minted.
- `to`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The receiver of tokens.
- `remoteToken`:
  - **Control**: Full control.
  - **Constraints**: The `localToken` address for this `remoteToken` cannot be zero.
  - **Impact**: The address of the token contract from the remote chain the tokens were sent from.
- `packetType`:
  - **Control**: Is not controlled.
  - **Constraints**: `packetType == PT_MINT`.

- **Impact**: The expected packet type. Means that tokens will be mint.

Function call analysis

- `IWrappedERC20(localToken).mint(to, amount)`
  - **What is controllable?** `to, amount`.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Can revert if the current contract cannot call the `mint` function - also if the `to` address is zero.

4.2 Module: `OriginalTokenBridge.sol`

Function: `registerToken()`

Allows the owner of the contract to register local tokens for transferring over the bridge.

Preconditions

The owner of the contract should be `msg.sender`. The token address shouldn't be registered before.

Branches and code coverage

Intended branches:

- The new non-zero token address will be registered successfully.
  ☑ Test coverage

Negative behavior:

- The `token` is already registered.
  ☑ Negative test?
- The caller is not the owner.
  ☑ Negative test?
- The `sharedDecimals` is more than `localDecimals`.
  ☑ Negative test?

Inputs

- `token`:
  - **Control**: Full control.
  - **Constraints**: Non-zero address.
  - **Impact**: Only registered tokens can be transferred over the bridge.
- `sharedDecimals`:
  - **Control**: Full control.
  - **Constraints**: `localDecimals ≥ sharedDecimals`.
  - **Impact**: The number of decimals used for all original tokens mapped to the same wrapped token.

Function: `setRemoteChainId()`

Allows the owner of the contract set the ID of the remote chain for token transfer.

Branches and code coverage

Intended branches:

- The new `remoteChainId` is set properly.
  ☑ Test coverage

Negative behavior:

- The caller is not the owner.
  ☑ Negative test?

Inputs

- `_remoteChainId`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Tokens from different `srcChainId` cannot be received.

Function: `bridge()`

Allows to transfer ERC20 tokens to `remoteChainId`. The sent tokens will be locked.

Preconditions

The `token` address is registered and `msg.sender` has enough amount of tokens.

Branches and code coverage

Intended branches:

- The token balance of the contract is increased by `amount` value.
  ☑ Test coverage
- The token balance of `msg.sender` is decreased by `amount` value.
  ☐ Test coverage
- Check that `amount` encoded inside the `payload` was converted to the wrapped tokens properly if `localDecimals == sharedDecimals`.
  ☐ Test coverage
- Check that `amount` encoded inside the `payload` was converted to the wrapped tokens properly if `localDecimals ≠ sharedDecimals`.
  ☑ Test coverage
- Check that `totalValueLockedSD[token]` was updated properly.
  ☑ Test coverage

Negative behavior:

- `token` address is not supported.
  ☑ Negative test?
- `msg.sender` doesn't have enough tokens.
  ☑ Negative test?

Inputs

- `adapterParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Value is used to determine the amount of produced gas limit.
- `callParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: refundAddress is the address of the receiver of fee surpluses - zroPaymentAddress if zero, fee in native tokens.
- `to`:

  - **Control**: Full control.
  - **Constraints**: `to` $\neq$ `address(0)`.
  - **Impact**: The address of the receiver of tokens on the remote chain side.
- `amountLD`:
  - **Control**: Full control.
  - **Constraints**: The caller has more or equal.
  - **Impact**: The number of tokens that the caller wants to transfer over the bridge.
- `token`:
  - **Control**: Full control.
  - **Constraints**: Should be `supportedTokens`.
  - **Impact**: Tokens that will be locked on the contract account.

Function call analysis

- `IERC20(token).safeTransferFrom(msg.sender, address(this), amountLD);`
  - **What is controllable?** `amountLD`.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The function is non-reentrant. Can revert if `msg.sender` doesn't have enough tokens.
- `_lzSend(remoteChainId, payload, callParams.refundAddress, callParams.zroPaymentAddress, adapterParams, nativeFee) -> lzEndpoint.send{value: _nativeFee}(_dstChainId, trustedRemote, _payload, _refundAddress, _zroPaymentAddress, _adapterParams)`
  - **What is controllable?** `refundAddress`, `zroPaymentAddress`, `adapterParams`, `nativeFee`, `payload` have full control.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Will revert if `remoteChainId` is unknown, if `msg.value` is not enough to pay the fee, or if `_zroPaymentAddress` address is not the contract address or is not `tx.origin` (in case of fee in ZRO token). Also, it will revert in case of error during refund of native tokens to `refundAddress`.

Function: `bridgeETH()`

The same as the `bridge` function, but the native tokens will be used for transferring.

Branches and code coverage

Intended branches:

- `Weth` balance of the contract increase by amount value.
  ☑ Test coverage
- Check that `amount` encoded inside the `payload` was converted to the wrapped tokens properly if `localDecimals == sharedDecimals`.
  ☑ Test coverage
- Check that `amount` encoded inside the `payload` was converted to the wrapped tokens properly if `localDecimals ≠ sharedDecimals`.
  ☐ Test coverage
- Check that `totalValueLockedSD[token]` was updated properly.
  ☑ Test coverage

Negative behavior:

- `to` is zero address.
  ☑ Negative test?
- `weth` address is not registered.
  ☑ Negative test?
- `msg.value` is less than amount value.
  ☑ Negative test?

Inputs

- `to`:
  - **Control**: Full control.
  - **Constraints**: `to ≠ address(0)`.
  - **Impact**: The address of the receiver of tokens on the remote chain side.
- `amountLD`:
  - **Control**: Full control.
  - **Constraints**: `msg.value ≥ amountLD`
  - **Impact**: The number of native tokens that the caller wants to transfer over the bridge.
- `adapterParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: Value is used to determine the amount of produced gas limit.
- `callParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.

- **Impact**: refundAddress is the address of the receiver of fee surpluses - zroPaymentAddress if zero, fee in native tokens.

Function call analysis

- `IWETH(weth).deposit{value: amountLD}();`
  - **What is controllable?** `amountLD`.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value here.
  - **What happens if it reverts, reenters, or does other unusual control flow?** No problems.
- `_lzSend(remoteChainId, payload, callParams.refundAddress, callParams.zroPaymentAddress, adapterParams, nativeFee)` -> `lzEndpoint.send{value: _nativeFee}(_dstChainId, trustedRemote, _payload, _refundAddress, _zroPaymentAddress, _adapterParams)`
  - **What is controllable?** `refundAddress`, `zroPaymentAddress`, `adapterParams`, `nativeFee`, and `payload` have full control.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Will revert if `remoteChainId` is unknown, if `msg.value` is not enough to pay the fee, or if `_zroPaymentAddress` address is not the contract address or is not `tx.origin` (in case of fee in ZRO token). Also, it will revert in case of error during refund of native tokens to `refundAddress`.

Function: `withdrawFee()`

Allows the contract owner to withdraw unlocked tokens as a fee.

Preconditions

The token balance of this contact is non-zero.

Branches and code coverage

Intended branches:

- The owner can withdraw fee properly.
  ☑ Test coverage

Negative behavior:

- The caller is not the owner.
  ☑ Negative test?
- The owner cannot withdraw locked funds.
  ☑ Negative test?

Inputs

- `amountLD`:
  - **Control**: Full control.
  - **Constraints**: amountLD <= fee.
  - **Impact**: The value that the caller wants to withdraw cannot be more than the unlocked amountLD of tokens.
- `to`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The receiver of tokens.
- `token`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The contract from which the caller wants to withdraw the fee.

Function call analysis

- `IERC20(token).safeTransfer(to, amountLD);`
  - **What is controllable?** `to`, `amountLD`, and `token`.
  - **If return value controllable, how is it used and how can it go wrong?** There isn't a return value here.
  - **What happens if it reverts, reenters, or does other unusual control flow?** No problem.

Function: `_nonblockingLzReceive()`

The part of the process of receiving tokens from another chain. Unlock the tokens and send to receiver. The input amount of tokens from payload data will be converted to the actual amount of tokens by proper `ConversionRate`.

Preconditions

The `totalValueLockedSD` for the token is a greater or equal `amount`.

Branches and code coverage

Intended branches:

- The balance of the `token` of the `to` address should increased by `withdrawalAmountLD` value in case of `token` != `weth`.
  ☑ Test coverage
- The receiver native token balance should increased by `withdrawalAmountLD` value in case `unwrapWeth` == true.
  ☑ Test coverage

Negative behavior:

- Incorrect packet type.
  ☑ Negative test?
- The `token` is not supported.
  ☐ Negative test?

Inputs

- `totalAmountSD`:
  - **Control**: Not controlled.
  - **Constraints**: Cannot be more than `totalValueLockedSD[token]`.
  - **Impact**: `withdrawalAmountSD` + fee.
- `withdrawalAmountSD`:
  - **Control**: Not controlled.
  - **Constraints**: Cannot be more than `totalValueLockedSD[token]`.
  - **Impact**: The number of tokens that the user receives (not including fee).
- `token`:
  - **Control**: Not controlled.
  - **Constraints**: `supportedTokens[token]` == true.
  - **Impact**: Address of the received tokens.
- `packetType`:
  - **Control**: Not controlled.
  - **Constraints**: packetType == PT_UNLOCK.
  - **Impact**: The expected packet type means that tokens will be unlocked.
- `srcChainId`:
  - **Control**: N/A.
  - **Constraints**: srcChainId == remoteChainId.
  - **Impact**: Tokens cannot be received from untrusted chain.

Function call analysis

- `(bool success, ) = payable(to).call{value: withdrawalAmount}("");`
  - **What is controllable?** `to`.
  - **If return value controllable, how is it used and how can it go wrong?** No problem.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Calling function is non-reentrant. In case of an error, native tokens receiving transaction will be saved for resending.

4.3 Module: `TokenBridgeBase.sol`

Function: `setUseCustomAdapterParams()`

Allows owner of bridge to set the `useCustomAdapterParams` flag. This flag indicates whether the contract should use custom adapter settings or default settings.

Branches and code coverage

Intended branches:

- The new value was set properly.
  ☑ Test coverage

Negative behavior:

- The caller is not the owner.
  ☑ Negative test?

Inputs

- `_useCustomAdapterParams`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: This value indicating whether the contract uses custom adapter parameters or the default ones.

Function: `_checkAdapterParams()`

Internal function. Allows to check gas limit if `useCustomAdapterParams` is set and verifies that caller's `adapterParams` value is zero otherwise.

Branches and code coverage

Intended branches:

- `_checkGasLimit` is successfully completed if `useCustomAdapterParams` is true.
  ☑ Test coverage

Negative behavior:

- `useCustomAdapterParams` is false and `adapterParams` is not zero.
  ☑ Negative test?

4.4 Module: `WrappedERC20.sol`

Function: `mint()`

Allows bridge contract `mint` new ERC20 tokens.

Branches and code coverage

Intended branches:

- The balance of `_to` address increased by `_amount` value.
  ☑ Test coverage

Negative behavior:

- The caller is not the bridge contract.
  ☑ Negative test?

Inputs

- `_to`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The receiver of new tokens.
- `_amount`:
  - **Control**: Full control.
  - **Constraints**: No checks.
  - **Impact**: The number of tokens to be minted.

Function: `burn()`

Allows bridge contract `burn` existing ERC20 tokens. Tokens must be minted before.

Branches and code coverage

Intended branches:

- The balance of `_from` address decreased by `_amount` value.
  ☑ Test coverage

Negative behavior:

- The caller is not the bridge contract.
  ☑ Negative test?

Inputs

- `_from`:
  - **Control**: Balance of `_from` must be more or equal to `_amount` value.
  - **Constraints**: No checks.
  - **Impact**: The owner of tokens that will be burned.
- `_amount`:
  - **Control**: Full control.
  - **Constraints**: Balance of `_from` must be more or equal to `_amount` value.
  - **Impact**: The number of tokens to be burned.

5 Audit Results

At the time of our audit, the code was not deployed to mainnet EVM.

During our audit, we discovered one low severity finding. LayerZero Labs acknowledged the finding and implemented a fix.

5.1 Disclaimers

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any additional code added to the assessed project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.